Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Магістерська робота

освітній ступінь – магістр

на тему: «**ОТРИМАННЯ ПРЕДСТАВЛЕНЬ ЗОБРАЖЕНЬ З УРАХУВАННЯМ ТЕКСТУ ДЛЯ ЗАДАЧІ ПОШУКУ НОМЕРНИХ ЗНАКІВ**»

Виконав: студент 2-го року навчання,

освітньо-наукової програми
«Прикладна математика», 113

Кольчик Микита Олегович

Керівник **Швай Н.О.**
к.н., доцент

Рецензент________________________
(прізвище та ініціали)

Магістерська робота захищена
з оцінкою ________________________

Секретар ЕК ______________________
«____» _________________ 20____ р.

Київ – 2024

КАЛЕНДАРНИЙ ПЛАН ПІДГОТОВКИ КВАЛІФІКАЦІЙНОЇ/МАГІСТЕРСЬКОЇ РОБОТИ ДО ЗАХИСТУ

Графік узгоджено «___» _________________ 20___ р.

№ з/п	***ПЕРЕЛІК РОБІТ***	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	жовтень			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	жовтень – листопад			
3.	Складання плану каліф. роботи та узгодження з науковим керівником	листопад			
4.	Написання розділів роботи	листопад – квітень			
5.	Проміжний контроль виконання роботи	лютий			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	січень – травень			

Вступ

Задача автоматичного розпізнавання номерних знаків (Automatic licence plate recognition, ALPR) є однією з найпопулярніших задач комп’ютерного зору. Цьому сприяє значне зростання попиту – камери спостереження з розпізнаванням номерних знаків та інші засоби безпеки.

Методи з розпізнавання об’єктів постійно вдосконалюються і мають все більшу точність, особливо такі, що побудовані на алгоритмах машинного навчання. Проте, більшість таких методів працюють за принципом одноразового визначення об’єкту, тобто, для одного і того самого об’єкту, але на різних зображеннях може відбуватися декілька операцій з розпізнавання. Це означає, що між зображеннями з однаковими об’єктами немає зв’язку, хоча очевидно, що це не так.

Зокрема, це стосується і номерних знаків. Було вирішено створити такий підхід, який би міг враховувати властивості зображень однакових номерних знаків. Такий підхід дозволив би робити пошук номерних знаків за запитом у вигляді текстової форми. Тому, завданням даної роботи є дослідження наявних алгоритмів з розпізнавання і пошуку номерних знаків, їх переваг та недоліків, а *метою* є створення алгоритму пошуку зображень номерних знаків за текстовим значенням. Наявні алгоритми мають певні недоліки- в основному за допомогою них не є можливим сформувати властивості набору зображень з однаковим номерним знаком, хоча очевидно, що такий набір матиме схожі характеристики.

Оскільки ідея такого алгоритму буде запропонована вперше, підхід щодо ототожнювання зображення з текстовою репрезентацією і буде науковою новизною. В даному алгоритмі текстовий запит перетворюється у синтетичне зображення, на якому навчається нейронна мережа, витягуючи основні характеристики реальних зображень з таким самим номером. Подібні методи використовуються у проблематиці “Image Similarity”, тому переосмисливши їх і підлаштувавши під нашу задачу, ми отримаємо оригінальний алгоритм.

Об’єкт дослідження - область Комп’ютерного Зору, яка стосується обробки зображень і розпізнавання номерних знаків.

Методи дослідження - методи розпізнавання тексту (EasyOCR), методи обробки зображень (OpenCV), методи глибокого навчання (Tensorflow)

Практичне застосування даного алгоритму може мати широке коло використання, а саме- задля контролю безпеки на дорогах (або контроль трафіку), трекінг автомобілів на стоянках з штучним інтелектом тощо.

Розділ 1 Місце згорткових нейронних мереж у детекції зображень

1.1 Згорткові нейронні мережі

На сьогодні, найкраще у роботі з детекцією і розпізнаванням зображень працюють згорткові нейронні мережі, майже всі state-of-the-art рішення побудовані на їх основі. Це стало можливо через їх гнучкість в архітектурі і здатності пристосування майже до будь-якої задачі. Вони використовуються для таких задач, як класифікація і сегментація зображень, розпізнавання об'єктів та інші.

Згорткова нейронна мережа (CNN) – один із видів нейронних мереж, які в основному спеціалізуються на аналізі візуальної інформації, що міститься на зображенні, також на аналізі часових і аудіо рядів.

Згорткова нейронна мережа діє по принципу виділення головних ознак зображення шляхом "згортки" самого зображення. Вона складається з шарів, які перетворюють зображення у вектор ознак, що дає змогу мережі навчитися на них. [17]

Загалом, найбільш розповсюджені шари нейронної мережі такі:

Згортковий шар (Convolution layer) – це шар, який дозволяє знайти певні ознаки у зображення шляхом накладання ядер у вигляді матриць, зазвичай розмірності 3x3. Після того, як виконується поматричне множення вхідного зображення на ядра, в результаті отримується так звана карта ознак, в якій містяться виділені ознаки, знайдені за допомогою ядер. Перші шари множення зображень на ядра дають більш широкі ознаки, в той час як останні – більш примітивні ознаки, такі, як лінії і кола.

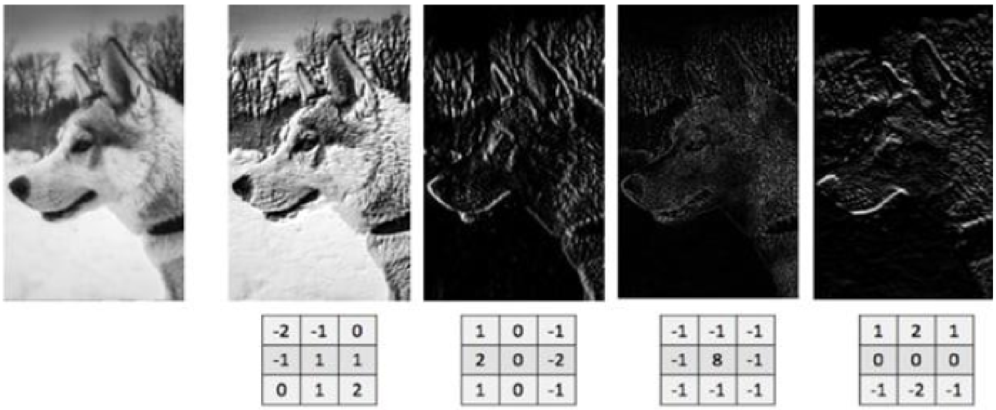

Рис.1.1 - візуалізація згорткового шару[1]

Пулінговий шар (Pooling layer) – шар, який зменшує розмірність карти ознак, зберігаючи найбільш значущу інформацію. Існують декілька методів, які застосовуються для зменшення розмірності, але найбільш популярний – MaxPooling. Такий шар бере найбільше значення пікселів з певної області, порівнюючи з сусідніми пікселями. Таким чином, зберігається вся необхідна інформація про ознаки і одночасно зменшується розмірність зображення в декілька разів, а це грає серйозну роль у швидкості навчання.

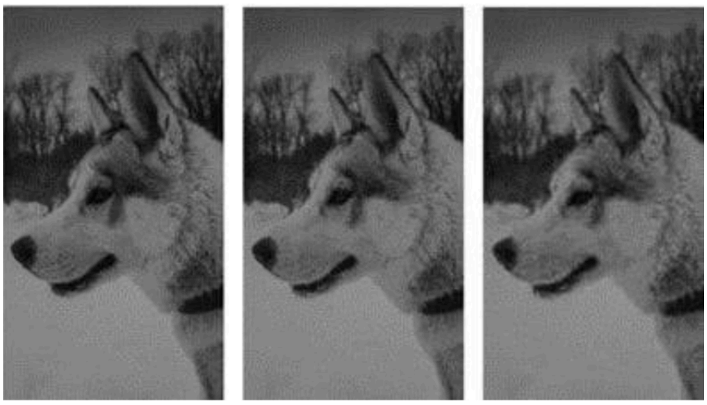

Рис.1.2 - візуалізація пулінгового шару[1]

Шар сплюснення (Flatten layer) – шар, який перетворює результати карти активацій у вектори для подальшого навчання нейронної мережі.

Тож, загальна архітектура згорткової мережі виглядає так:

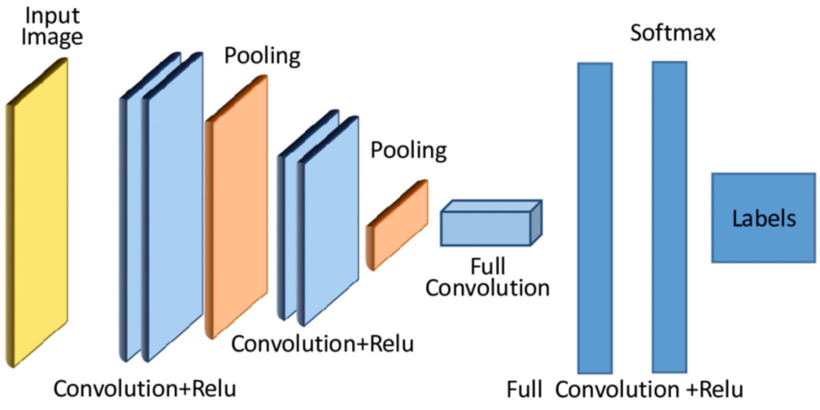

Рис.1.3 - візуалізація архітектури згорткової мережі[2]

1.2 Автокодувальники

Автокодувальник – алгоритм навчання без вчителя, робота якого – повернути зображення, ідентичне поданому на вхід.

Мета автокодувальника – навчитися виявляти ознаки на високорозмірних даних таким чином, щоб максимально точно відтворити вхідне зображення. Результатом роботи є згенероване синтетичне зображення, яке повинно відповідати результатам задачі. Існує декілька задач, які вирішує автокодувальник:

- Зменшення шуму на зображенні. Подаються два датасети - На вхід зашумлені зображення, а на виході – зображення без шуму. Мережа вчиться на основі таких двох датасетів генерувати зображення без шуму.
- Доповнення області на зображенні, якої бракує. Подаються повні зображення і зображення з деякою закритою областю. Задача мережі – згенерувати заміщення закритої області згідно з контекстом зображення.
- Генерування схожих зображень. Податься два датасети зі схожими зображеннями, і задача мережі – згенерувати зображення з одного

датасету, що максимально схоже на зображення з другого. Така задача ідеально підходить під нашу роботу, тому саме вона буде взята за основу.

1.3 Архітектура автоавтокодувальників

Базова архітектура автокодувальника складається з трьох частин:

- Енкодер – згорткова нейронна мережа (CNN), яка стискає вхідні зображення у вектор ознак, будуючи одномірне представлення для ознак кожного зображення.
- Латентний простір – місце, де зберігаються репрезентації стиснутих зображень у вигляді векторів
- Декодер – нейронна мережа, яка є оберненою до енкодера і будує зображення на основі репрезентацій з латентного простору. Ця частина автокодувальника намагається з найбільшою точністю відтворити вхідні дані.

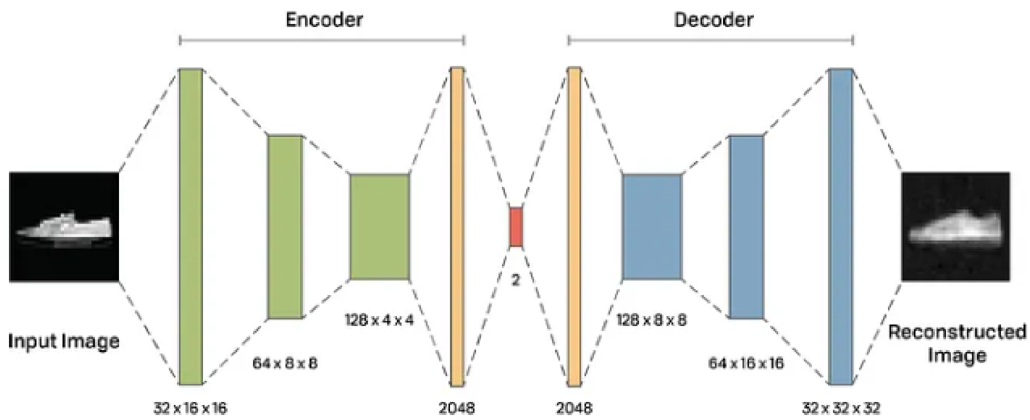

Рис.1.4 - Архітектура автокодувальника[3]

1.4 Види автокодувальників

Найбільш поширеними версіями автокодувальників є: звичайний автокодувальник (vanilla autoencoder), варіаційний автокодувальник (variational autoencoder) і автокодувальник, що зменшує шум (denoising autoencoder) [18]

Особливою відмінністю варіаційного автокодувальника від ванільного є те, що другий створює один вектор для навчання, в той час як варіаційний два – вектор середніх значень і вектор середніх відхилень. Це дає автокодувальнику більше можливостей для навчання, бо такий енкодер зосереджений не на одній точці, а на області, яка схожа на вхідний сигнал.

1.5 Різниця між ванільними і варіаційними автокодувальниками

Враховуючи попереднє, можна зробити висновки, що такі два види автокодувальників відрізнятимуться функцією втрат. Нехай x – вхідний сигнал, $\hat{x}$ – згенерований сигнал. Тоді функція втрат виглядатиме таким чином:

$$l = \left|\left|x - \hat{x}\right|\right|_2 = \left|\left|x - d_{\varphi}\left(e_{\theta}(x)\right)\right|\right|_2$$

де e_{θ}- функція енкодера, d_{φ}- функція декодера.

В той же час, функція втрат варіаційного автокодувальника складається з двох частин – функція втрат реконструкції і функція втрат схожості. І глобальна функція втрат є сумою таких двох функцій втрат. [11]

$$l_{reconstruction} = \left|\left|x - \hat{x}\right|\right|_2 = \left|\left|x - d_{\varphi}(z)\right|\right|_2 = \left|\left|x - d_{\varphi}(\mu_x + \sigma_x\varepsilon)\right|\right|_2, \text{ де}$$

$$\varepsilon = N(0, 1)$$

$$l_{similarity} = D_{KL}(N\left(\mu_x, \sigma_x\right)||N(0, 1), \text{ де}$$

$D_{KL} -$ розходження Кульбака $-$ Лейблера $(Kullback - Leibler\ divergence)$

$$loss = l_{reconstruction} + l_{similarity}$$

Функція втрат реконструкції намагається відновити сигнал, що був поданий на вхід. Функція втрат схожості намагається зробити латентний простір нормально розподіленим.

1.6 Імплементація автокодувальника у Tensorflow

TensorFlow — це фреймворк машинного навчання з відкритим кодом, розроблений Google. Він був розроблений, щоб полегшити створення моделей машинного навчання, зокрема моделей глибокого навчання, для різноманітних завдань, таких як розпізнавання зображень, обробка природньої мови, детекції зображень тощо.. TensorFlow надає гнучку та ефективну систему для створення та навчання нейронних мереж, дозволяючи користуватися їхнім API. Зокрема модуль Keras, є частиною TensorFlow, являє собою високорівневий інтерфейс для побудови та навчання моделей штучних нейронних мереж. Його зручність полягає в тому, що він дозволяє швидко створювати складні моделі, використовуючи такі шари, як Dense, Conv2D, та LSTM. При цьому, є можливість використати GPU для прискорення навчання таких моделей. Як показують дослідження,

1.7 Пошук схожих зображень

Для початку варто визначити, що означає «схожість» двох зображень. Можна виділити два типи схожості. Перший тип – схожість за порівнянням контрастності пікселів. Тобто, використовуються методи порівняння гістограм кольорів, відстань між кольорами, структурні характеристики зображень тощо. Другий – схожість за контекстом. Наприклад, якщо на двох зображеннях є коти, а на третьому – собака, то два перших зображення будуть контекстуально більш схожими, аніж перше і друге з третім. Така проблема існує у задачі класифікації, коли алгоритму потрібно навчитися на наборі зображень віднести зображення до певних класів.

Результатом навчання нейронної мережі (автокодувальника) є спільний простір у якому містяться представлення вхідних і вихідних зображень. Використовуючи цей простір, можна знайти зображення, максимально схожі на згенероване (синтетичне). Представлення зображень – вихідні вектори, навчені автокодувальником, і такі вектори мають певні властивості, а саме –

більш схожі зображення розташовані у просторі ближче одне до одного, ніж менш схожі. Чому так відбувається? Параметри нейронної мережі змінюються таким чином, щоб мінімізувати розходження між згенерованим і вхідним зображенням і, відповідно, схожі зображення будуть розташовані поруч в N-мірному просторі.

Для простої візуалізації схожості зображень у просторі, можна проілюструвати це на прикладі датасету MNIST, попередньо зменшивши розмірність вектору до двовимірного, щоб показати це на площині.

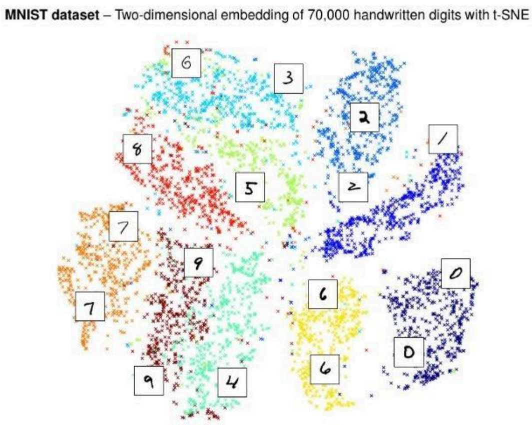

Рис.1.5 - візуалізація згорткового шару[4]

Отже, можна зробити висновок, що схожість зображень це відстань між їх векторами ознак. Яким чином вимірюється відстань між векторами ознак? Є декілька методів, що її вимірюють, найпоширеніші – евклідова відстань (euclidian distance) і косинусна подібність (cosine similarity).

Евклідова відстань –сума квадратів різниць координат векторів у просторі.

$$d = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

Тоді подібність між векторами буде обернено- пропорційна до відстані:

$$s = \frac{1}{1+d}$$

Косинусна подібність – величина, що вимірює кут між векторами і лежить в проміжку $[-\ 1; 1]$. Якщо задані два вектори ознак A і B, то косинусна подібність обчислюється таким чином:

$$s = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}}$$

1.8 Сіамські нейронні мережі для пошуку схожих зображень

Такі мережі мають особливу архітектуру, яка складається з двох однакових нейронних мереж, які приймають на вхід пари зображень. На виході отримуються векторні репрезентації, які потім порівнюються для того, щоб визначити подібність між зображеннями. Суть таких мереж в тому, щоб навчити мережі представляти зображення у векторах так, щоб розуміти основні характеристики схожості на парах зображень. Особливістю є те, що на вхід можуть подаватися не розмічені зображення, тобто без класів. Сіамська мережа складається з таких компонентів: дві однакові згорткові мережі, метрики подібності і контрастної функції втрат. Оскільки дві мережі мають спільні ваги, це даж можливість ефективно знаходити спільні ознаки для обох зображень. Метрика подібності використовується для порівняння виходів у вигляді векторів. Зазвичай, найпопулярнішими метриками є косинусна подібність і евклідова відстань. Контрастна функція втрат необхідна для того, щоб штрафувати мережу, коли подібність вища за деякий поріг. При мінімізації контрастної функції втрат, мережа вчиться ототожнювати векторні представлення обох зображень, фіксуючи найбільш схожі спільні характеристики зображень. Варто зауважити, що контрастна функція втрат відрізняється від інших тим, що для неї на вхід подаються не тільки схожі зображення, а і не схожі. Це означає, що схожі зображення матимуть одну мітку, а не схожі- іншу, для розрізнення таких пар. Формула для функції втрат виглядає так [12]:

$$L(W, (Y, X_1, X_2)^i) = (1 - Y)L_s(D_W{}^i) + YL_D(D_W{}^i) \quad \text{, де}$$

$$L_s - \text{функція втрат для схожих зображень,}$$

$$L_d - \text{функція втрат для не схожих зображень,}$$

$$D^i_W - \text{ метрика схожості для пари векторів,}$$

Оскільки використовується слово "схожість", вона вимірюється у відсотках, у нашому випадку це показник у проміжку $[0, 1]$. Тобто, евклідова відстань або косинусна схожість пропускаєтсья через сігмоїдальну функію, і отримується відповідне значення. Нижче на малюнку зображено загальну архітектуру:

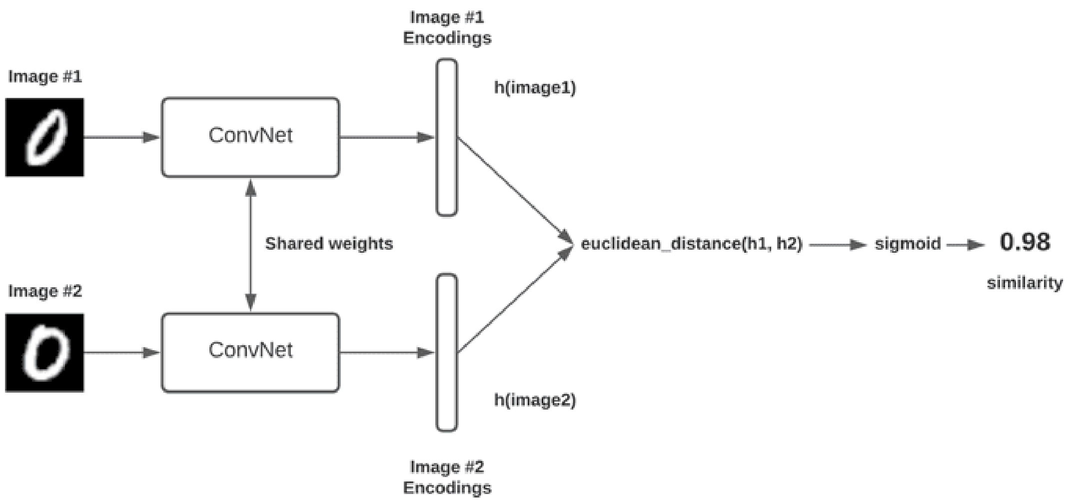

Рис.1.6 - Загальна архітектура сіамської мережі[5]

1.9 Висновки до розділу 1

Нейронні мережі є потужними алгоритмами для роботи із зображеннями, і їх можна використати для великої кількості оригінальних задач. Зокрема, автокодувальники можна застосувати як інструмент відтворення зображень, що, в свою чергу, відкриває широкі можливості по створенню програм з ототожнювання зображень, що знадобиться нам в цій роботі. Це дає змогу порівнювати пари зображень, що подаються на вхід, і навчити мережу подавати на вихід схожі зображення з бази даних векторів ознак.

Вектори ознак (ембеддінги) є результатом навчання нейронної мережі, і кожне зображення з набору даних матиме свій унікальний вектор, і схожі зображення у цьому наборі матимуть ближчі вектори, ніж не схожі зображення.

Розділ 2 Наявні методи розпізнавання номерних знаків.

2.1 OCR методи з розпізнавання тексту

OCR (Optical Character Recognition) - технологія, що дозволяє перетворювати зображення, на якому міститься текст у текстовий формат. Першими методами з розпізнавання тексту були методи на основі пошуку шаблонів (template matching). Тобто, для кожного символу існує шаблон, і шляхом посимвольного порівняння знаходиться відповідник до вхідного символу на зображенні. Зі зростанням популярності машинного навчання, з'явилася велика кількість методів OCR, які побудовані на базі цієї технології. Очевидним було рішення застосувати згорткові нейронні мережі, такі, як VGG-16, адже саме вони найкраще працюють із обробкою зображень. Перші роботи по втіленню методів OCR на основі нейронних мереж мали дві нейронні мережі - перша відповідала за пошук області з текстом на зображенні (Text Localizer), а друга мережа відповідала за визначення зображеного тексту (Text Detector). Згодом, такі алгоритми були вдосконалені до таких, що вирішують задачу розпізнавання тексту на дуже високому рівні.

Бібліотека EasyOCR - одне з таких рішень. Це модуль на мові python, який розпізнає текст на зображенні [10]. Фреймворк складається з багатьох частин, проте основні дві: перша - CRAFT (Character Region Awareness for Text Detection) - частина, що відповідає за знаходження тексту і виділення області з текстом, а друга - за знаходження ознак (ResNet) і декодування символів (LSTM).

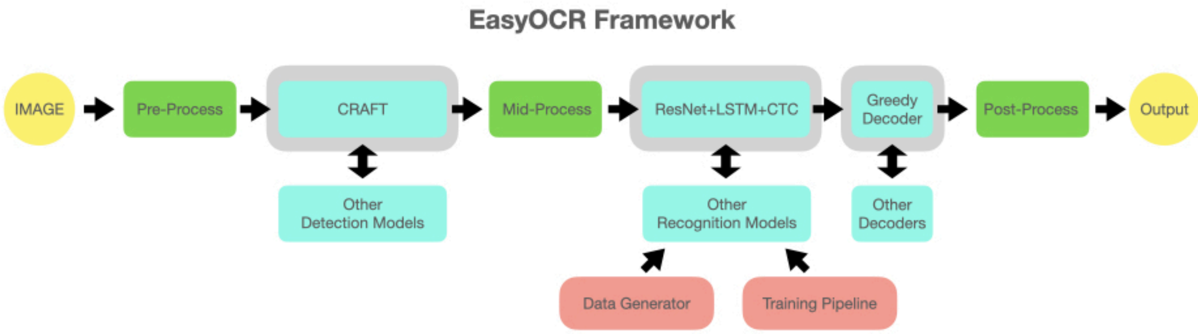

Рис.2.1 - Фреймворк EasyOCR[6]

2.2 Переваги розпізнавання з OCR

Методи, які працюють за подібною логікою, можуть чітко розпізнавати окремі символи за допомогою переднавчених алгоритмів машинного навчання, оскільки такі алгоритми з точністю більше, ніж 90% розпізнають символи коректно.

По- друге, це різноманітність таких бібліотек, адже окрім EasyOCR існує ще більше схожих, такі, як Pytesseract, Keras-OCR, Doctr та інші.

По-третє, алгоритми як EasyOcr можуть виділяти області зі знайденим текстом прямокутником, який можна вирізати з зображення і використовувати його замість першого, що дозволить зменшити шум і зберегти тільки необхідну область.

2.3 Недоліки розпізнавання з OCR

Проте, такі алгоритми діють доволі наївним чином, адже вони дозволяють розпізнавати тільки номерні знаки без прив'язки до самого номера. Мається на увазі те, що маючи декілька зображень з однаковим номерним знаком, ці зображення будуть розпізнаватися окремо і не будуть пов'язані між собою ніякими властивостями, що є не зовсім так. Очевидно, що зображення зі спільним номерним знаком матимуть деякі схожі властивості. Або ж, алгоритми для навчання можна підлаштувати так, щоб він відповідав вимогам нашої задачі.

Другим недоліком є те, що цей підхід працює тільки в одну сторону. Тобто, визначити текстовий формат номера можливо за зображенням, а знайти зображення (або декілька) по тексту номерного знака неможливо.

2.4 Найбільш використовуваний метод розпізнавання номерних знаків

Зі збільшенням кількості і якості алгоритмів комп’ютерного зору, розпізнавання номерних знаків стало можливістю для тестів різних методів. Проаналізувавши достатню кількість статей і публікацій, можна зробити висновок: зазвичай, робота по розпізнаванню номерних знаків поділяється на кілька етапів [13,14,15,16] :

1) Детекція номерного знаку
2) Обробка отриманого зображення номерного знаку
3) Сегментація символів на обробленому зображенні
4) Посимвольне розпізнавання літери (цифри)

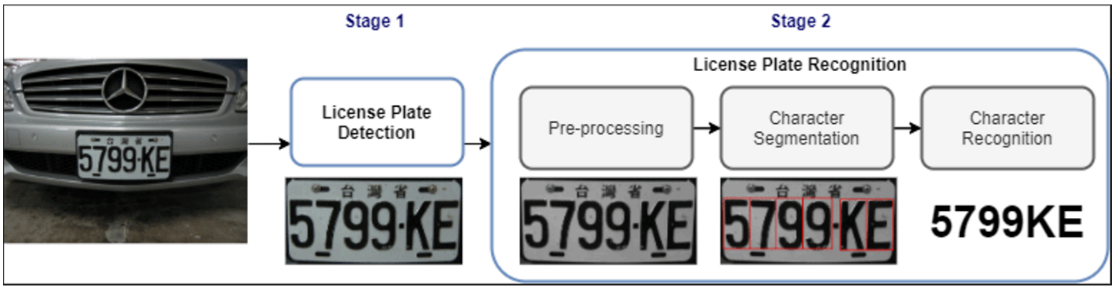

Рис.2.2 - Шаблонний метод розпізнавання номерних знаків[7]

Розглянемо кожен етап окремо.

Детекція області з номерним знаком відбувається за допомогою переднавчених алгоритмів машинного навчання, наприклад, YOLO. YOLO (You Only Look Once)- алгоритм з області object detection, що дозволяє знаходити області з необхідними об’єктами на зображенні. Його доволі просто навчити на власному наборі даних, тому це є оптимальним вибором у задачі знаходження області з номерними знаками. Цей крок є необхідним для того, аби позбутися непотрібних елементів на зображенні і прибрати усе, окрім номерного знаку. Тобто, за допомогою цього кроку покращується якість вхідних зображень.

Обробка отриманого зображення відбувається на основі бібліотеки OpenCV, яка має велику кількість функцій для зменшення зашумленості у вхідних зображеннях. Для цього використовуються функції розмиття (GaussianBlur) і використання порогу для перетворення зображення у двох кольорове - чорне і біле (Threshold). Після застосування таких перетворень, зображення обробляються алгоритмами по розпізнаванню тексту.

Посимвольне розпізнавання тексту на обробленому зображенні відбувається за допомогою білліотек, таких як EasyOCR та інших.

Такий підхід є однозначно швидким і надійним, адже з бібліотеками машинного навчання можливо з досить великою точністю визначити символи на зображенні.

2.5 Висновки до розділу 2

В даному розділі були розглянуті методи з розпізнавання номерних знаків на основі бібліотек машинного навчання, таких як EasyOcr. Ці методи мають популярність через швидкість і якість розпізнавання символів на зображенні, проте, мають ряд недоліків, які не дозволяють робити пошук на не розмічених даних.

Водночас, більшість авторів різних статей на цю тему використовують саме такі алгоритми, адже вони мають на меті трохи іншу задачу. Для нашої задачі алгоритми OCR використовуватимуться лише для виділення області з необхідним знаком, а для самого розпізнавання взятий за основу інший підхід.

Розділ 3 Алгоритм співставлення зображень номерних знаків

3.1 Опис

Перед нами стояла нетривіальна задача – навчити нейронну мережу розпізнавати номерні знаки так, щоб вони відповідали текстовій репрезентації. Питання – чи можливо це зробити наявними методами? На перший погляд, можливим було застосувати state-of-the-art фреймворк CLIP від OpenAI, який здатний пов'язувати зображення із його текстовою репрезентацією. Він навчений розуміти контекст зображення і видавати його словесний опис. CLIP побудований на основі декількох нейронних мереж, які взаємодіють між собою. Одна нейронна мережа (наприклад ResNet) приймає на вхід зображення і видає векторну репрезентацію для нього. Інша мережа працює з текстом (наприклад Text Transformer) і видає векторну репрезентацію (embedding) для тексту. Однак, ця мережа, як і всі схожі на неї, працює з реченнями і словами, розуміючи суть самого речення і слів у ньому, в той час як текстовий номер – це послідовність не пов'язаних між собою символів, і така послідовність не має ані контексту, ані коренів слів, що створює певну проблему в роботі з цим фреймворком, адже його буде неможливо використати на текстових репрезентаціях номерних знаків.

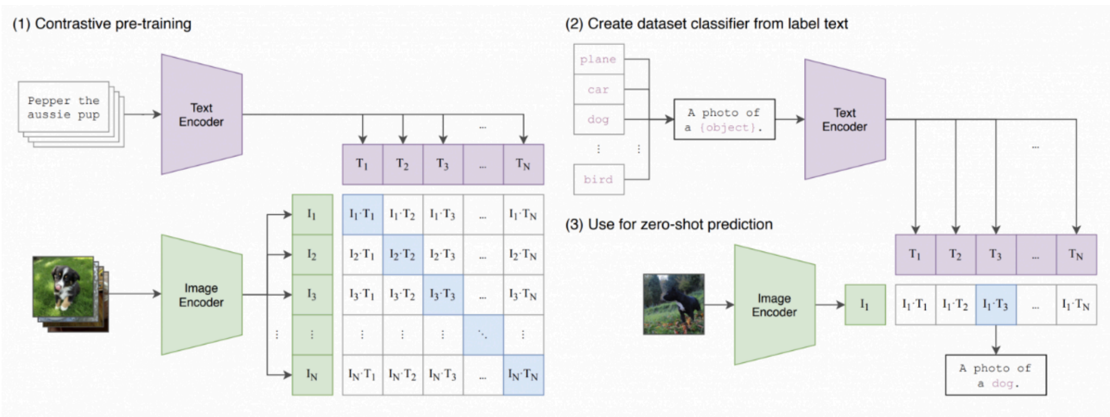

Рис.3.1 - CLIP від OpenAI[8]

Тому, було вирішено підлаштувати задачу під більш підходящу, у якій стало б можливим використати наявні нейронні мережі для навчання лише використовуючи зображення, тобто – перейти від нейронних мереж, що спеціалізуються на парі «зображення-текст» до «зображення-зображення».

Алгоритм повинен вирішувати такі задачі:

1) Розпізнавання номерного знаку
2) Пошук зображень з ідентичним номерним знаком
3) Пошук зображень за текстовою репрезентацією номеру з ідентичним номером.

3.2 Знаходження номеру

Оскільки датасет містить велику кількість зображень, необхідно було швидко і ефективно використати детекцію номерного знаку на зображенні таким чином, щоб позбавити зображення зайвого шуму, який не відноситься до номерного знаку (частіше за все це елементи кузова автомобіля).

Для цього було вирішено використати бібліотеку EasyOCR на основі алгоритів машинного навчання. За допомогою неї стало можливо швидко виділити автомобільні номери і вирізати їх у нові зображення, замінивши початковий датасет і покращивши якість вхідних зображень.

Цей крок допоміг виділити область, яка буде обмежувати визначені символи знака. Після цього, область зі знайденим текстом вирізається з зображення і зберігається замість наявного. Таким чином, стало можливо покращити якість зображень для подальшого навчання.

3.3 Обробка датасету

Перед тим, як тренувати датасет, необхідно було обробити зображення таким чином, щоб максимально виділити символи на знаках. Для цього була використана бібліотека OpenCV, яка має велику кількість методів обробки зображень. Тому за допомогою цієї бібліотеки зображення оброблялись у такій послідовності:

1) Перший крок - перетворення зображення в чорно-білий формат (grayscale) для того, щоб працювати з масивами розмірності 2, а не 3, оскільки кольорові зображення мають таку розмірність через три основних кольори червоний, зелений, синій (RGB)
2) Другий крок - Застосування білатерального фільтру (BilateralFilter) для зменшення зашумленості зображення. білатеральний фільтр розмиває зображення і, відповідно, зменшується кількість зайвих пікселів, які могли би вплинути на навчання.
3) Третій крок - Застосування порогу (thresholding), який перетворив зображення на набір пікселів або білого або чорного кольору. Таким чином отримується зображення, яке має символи чорного кольору, а фон - білого кольору.
4) Четвертий крок - Зміна розмірів зображення до формату 512x64 пікселі для уніфікації всіх зображень, адже початковий набір даних має зображення різної розмірності.

Такі перетворення дали змогу максимально виокремити символи на номері, для покращення навчання і зменшення функції втрат. В результаті, зображення позбулися шуму, кольорів і текст на них став набагато більш чітким:

Рис.3.2 - Результат обробки зображення

3.4 Генерація синтетичних зображень

Було вирішено відійти від роботи з текстовими репрезентаціями для того, щоб працювати виключно у площині зображень. Вирішенням цієї проблеми стала генерація синтетичних зображень для кожної текстової репрезентації. Для кожної такої генерується відповідне зображення із зображень літер і цифр стандартного шрифту автомобільних номерів. Такий крок дозволив нам відійти від порівняння зображень і тексту, а напряму працювати тільки із зображеннями. Для нашого датасету, в якому присутні декілька зображень з однаковим номером, це стало великою перевагою, адже автокодувальнику стало легше знаходити закономірності між зображеннями, що мають однаковий сенс (в нашому випадку однаковий номер).

Усі такі синтетичні зображення матимуть розмірність 512x64 пікселів, як і оригінальні зображення.

Приклади синтетичних зображень:

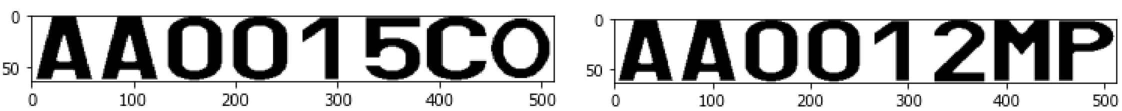

Рис.3.3 - Приклад синтетичного зображення

Оскільки датасет було вже розмічено, тобто кожне зображення було підписане відповідним номером, можна співставити реальне і синтетичне зображення, оскільки зображення з датасету реальних зображень матиме такий самий індекс, як і синтетичне.

Генерація таких зображень відбувалась по наступному принципу: для кожного розміченого зображення в папці збирається текстове значення (тобто, назва зображення), надалі воно перетворюється в масив довжиною кількості символів у назві; для кожного символа створюється зображення з літерою або цифрою з алфавіту, і всі ці зображення об’єднуються в одне, як було вказано раніше, розмірністю 512x64. Це дозволило уніфікувати всі номерні знаки, незалежно від кількості символів, адже в різних датасетах вони мають різну кількість символів. Також, це необхідно для подальшого навчання нейронною мережею через те, що всі вхідні зображення повинні бути одного розміру.

3.5 Використання автокодувальника

Ідея запропонованого підходу полягає в тому, щоб створити таку програму, яка була б універсальною і виконувала задачу співставлення синтетичного і реального зображення. Як це повинно працювати? Оскільки ми хочемо знаходити як зображення по текстовому номеру, так і текстовий номер на зображенні, необхідно побудувати таку модель, щоб текстова репрезентація і відповідне зображення були якимось чином пов'язані. Це можливо зробити за допомогою автокодувальника, який вчиться видавати згенероване зображення, маючи вхідне зображення.

Суть синтетичного зображення для автокодувальника – стати «ідеальним» зображенням автомобільного номеру, яке автокодувальник повинен генерувати для реальних фотографій з таким самим номером. Тобто, маючи декілька зображень з однаковим номером, повинен отримуватись один і той самий результат.

Зазвичай, автокодувальник використовується для відтворення на базі одного зображення, де вихідний сигнал повинен відповідати вхідному сигналу, але було вирішено видозмінити його суть, де на вхід подається реальне зображення, а на вихід- синтетичне, і автокодувальник відтворює не просто вхідне зображення, а знаходить залежності між першим і другим, видаючи результатом репрезентацію вхідного зображення у просторі синтетичних, що є теоретичним розв'язком нашої задачі.

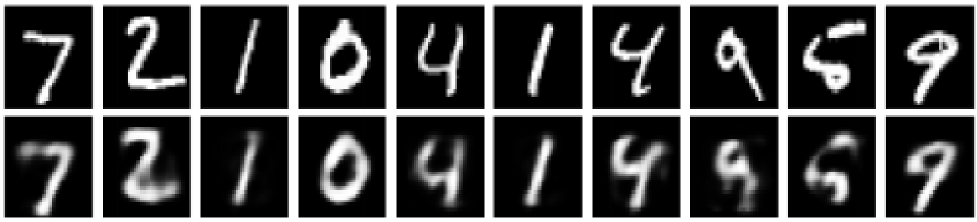

Рис.3.4 - Приклад відтворення зображення автокодувальником[9]

3.6 Архітектура обраного автокодувальника

Тренування автокодувальника було вирішено зробити за допомогою бібліотеки Tensorflow, яка підходить під нашу задачу. В ній містяться шари для створення автокодувальника, який, як було вказано раніше, складається із зготркових шарів і спочатку стискає вхідне зображення, а потім розтискає його у ту ж саму розмірність.

Також, було вирішено не використовувати складні автокодувальники накшталт варіаційного через надання переваги глибшій архітектурі ванільному автокодувальнику. Причиною на це був довгий час тренування, через що було вирішено зробити таким чином.

Архітектура обраного автокодувальника складається з 7 шарів згорток (Conv2D) і 6 шарів MaxPooling2D, що забезпечує достатню, на нашу думку, розмірність для обрахунку оптимальних векторів ознак. Початковий шар згорткової мережі було обрано розмірності 1024, а кінцевий- розмірності 8. Тож, архітектура енкодера виглядає так:

Conv2D(1024)→MaxPooling2D((2,2))→Conv2D(512)→MaxPooling2D((2, 2))→Conv2D(256)→MaxPooling2D((2,2))→Conv2D(64)→MaxPooling2D((2, 2))→Conv2D(32,(3,3))→MaxPooling2D((2,2))→Conv2D(16,(3,3))→MaxPooling 2D((2, 2))→Conv2D(8, (3, 3))→MaxPooling2D((2, 2))

При розгортці стиснутого зображення використовується функція, зворотня до MaxPooling2D - UpSampling2D, тому для отримання вихідного зображення необхідно також замінити один шар на інший, тому архітектура декодера виглядає так:

Conv2D(8)UpSampling2D((2,2))→Conv2D(16)→UpSampling2D((2, 2))→Conv2D(32)→UpSampling2D((2,2))→Conv2D(64)→UpSampling2D((2, 2))→Conv2D(128,(3,3))→UpSampling2D((2,2))→Conv2D(256,(3,3))→UpSampling2D((2, 2))→Conv2D(512, (3, 3))→UpSampling2D((2, 2))

3.7 Навчання автокодувальника

Отже, ми маємо два датасети – датасет, що містить оброблені зображення і датасет (X), що містить синтетичні зображення (Y). Ці датасети використовуватимуться для навчання автокодувальника. На вхід до автокодувальника подається пара зображень X[i], Y[i], як на прикладі нижче:

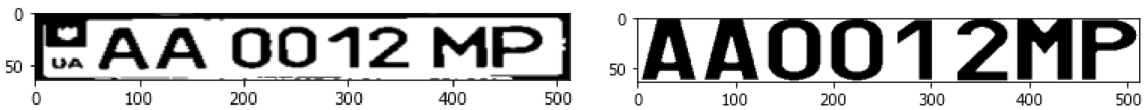

Рис.3.4 - Приклад пари зображень, що подається на вхід

Як було вказано раніше – задача автокодувальника є генерування зображення на основі датасету реальних зображень, що будуть максимально відповідати зображенням синтетичним.

Зазвичай, автокодувальник тренується функцією autoencoder.fit(X,X) для отримання згенерованого зображення, проте, оскільки маємо іншу задачу, використовуються інші параметри: autoencoder.fit(X, Y)

Функцією втрат обрано Mean Squared Error (MSE), а метрикою оцінки якості навчання - accuracy.

Загалом, навчання автокодувальника зайняло близько 2 годин, алгоритм навчався впродовж 500 епох. Така кількість епох дозволила максимально зменшити помилку навчання і збільшити accuracy до 98%. Завдяки такому високому показнику, стало можливим точно створити векторні асоціації до зображень.

3.8 Висновки до розділу 3

У розділі 3 було запропоновано алгоритм, який має на меті асоціювати “ідеальні” зображення номерного знаку, що були отримані з текстового запиту з зображеннями із набору даних. Ідея полягала в тому, що за допомогою автокодувальника можливо зробити таке ототожнення, навчивши його на парах синтетичного і реального зображень. Як можна буде побачити нижче, ця гіпотеза спрацює, адже алгоритм навчиться створювати такі вектори ознак, що дадуть змогу знайти за схожими векторами схожі зображення з набору даних. Навчання було ускладнене тим, що самі зображення є доволі великими (зазвичай, тренують на зображеннях розмірності 28x28 пікселів), і тому було вирішено використати просту архітектуру ванільного автокодувальника, не використовуючи складніші архітектури.

Розділ 4 Результат навчання автокодувальника

4.1 Тренування на різних вхідних даних

Спочатку варто зауважити, що навчання на обох датасетах відбувалось за допомогою GPU для того, щоб прискорити процес навчання.

Для навчання автокодувальника і тестування його роботи було використано два датасети. Перший – містить 370 зображень чеських номерних знаків і кожне зображення містить унікальний номер. Другий датасет – складається із 60 тисяч зображень, в ньому міститься зображення, які мають не унікальні номерні знаки, тобто такі, що повторюються, але зроблені з різних ракурсів.

Перший датасет використовували для підтвердження, що такий підхід валідний і його можна масштабувати, навчивши мережу на більшому датасеті. Оскільки перший датасет мав досить малу кількість зображень, результат навчання на ньому показав доволі посередні результати.

4.2 Результат навчання на першому наборі даних

Результат навчання: По-перше, тестовий набір даних пропускається через модель і отримуються векторні представлення для кожного реального зображення. Далі, генеруються номерні представлення, які відповідають відповідному зображенню.

Оскільки в першому наборі міститься доволі мала кількість даних, було очевидно, що високі результати навчання не очікувалось через те, що всі зображення в ньому унікальні. Проте, такий тест повинен був показати, чи валідний такий підхід до роботи за таким принципом із зображеннями взагалі. Виявилось, що навіть на невеликому датасеті, результати дали оптимістичні висновки. По-перше, автокодувальник генерував зображення, на яких декілька символів із нього номеру були правильними. Приклад згенерованого зображення показано нижче:

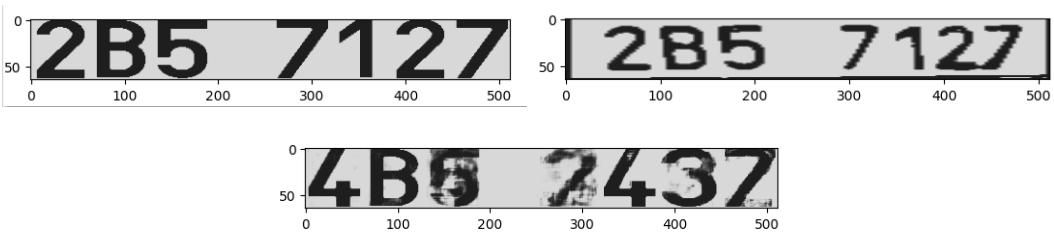

Рис.4.1-4.3 - Синтетичне, реальне і згенероване зображення

Як можна побачити, навіть на невеликому наборі даних алгоритм генерує зображення, де три з семи символів розпізнаються правильно, а це може означати, що на великому датасеті якість відтворення вхідного зображення має бути кращою.

4.3 Результат навчання на другому наборі даних

Основна метрика, на яку була звернута увага - accuracy. Як можна побачити, протягом 500 епох, вона збільшувалась. В результаті, кінцеве значення метрики - 0.98, Що означає хороший результат навчання.

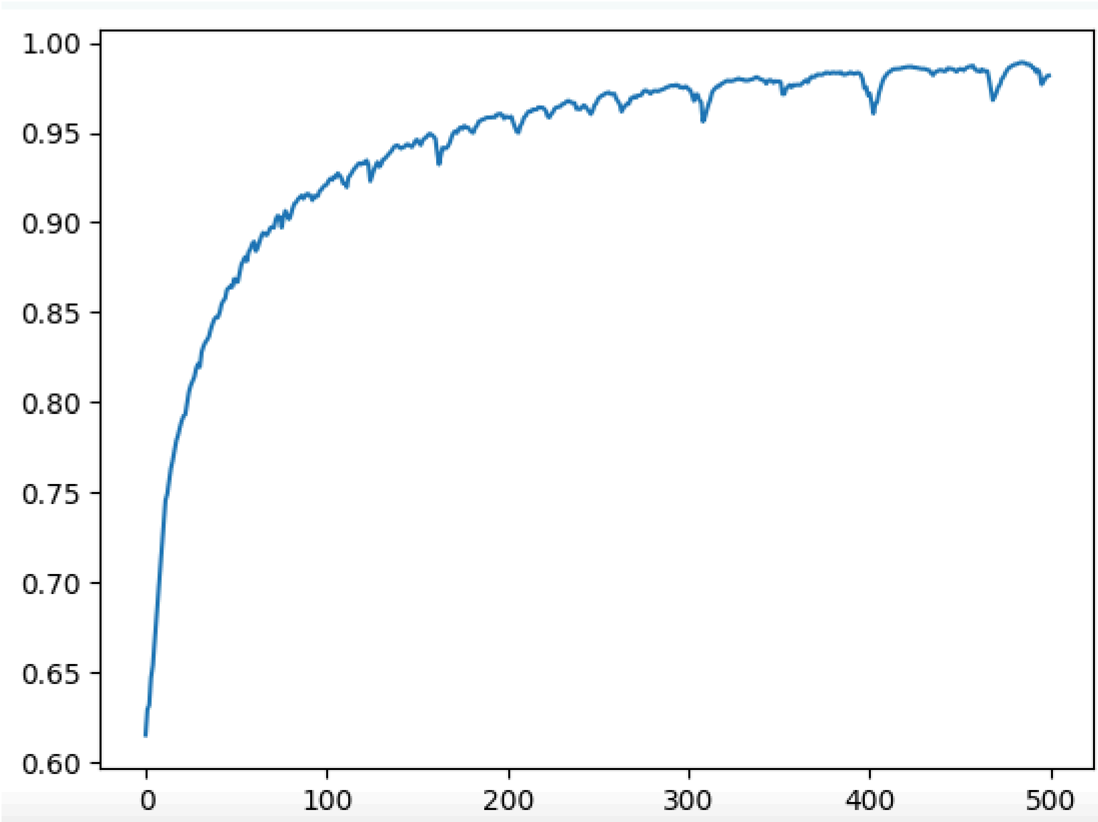

Рис.4.4 - Графік результату зростання метрики accuracy

4.4 Пошук зображень за текстом

Усі зображення з одним номером відповідають одному і тому же синтетичному зображенню. Отже, автокодувальник ідентифікує всі такі зображення з одним вектором ознак і ідеально, вони повинні бути однакові. При пошуку зображень за номером (текстовим значенням), виводитись повинні ті зображення, що відповідають такому номеру. Приклад зображено на рисунку нижче:

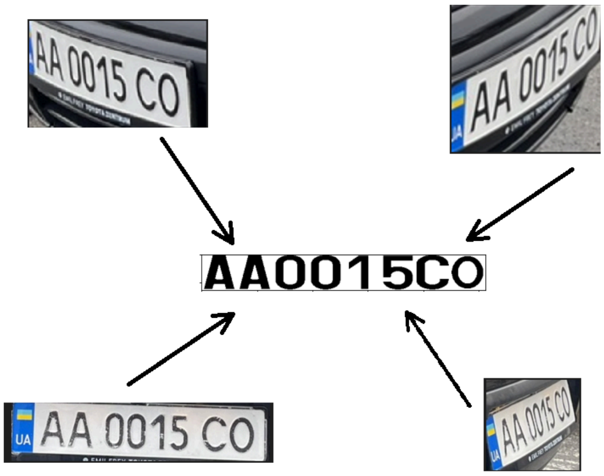

Рис.4.5 - Приклад асоціації текстового запиту і набору зображень

Тобто, після навчання автокодувальника ми отримуємо вектори ознак для реальних і синтетичних зображень, і тому ми можемо їх порівняти. Це дає змогу визначити, які зображення найбільш близькі до обраного номеру.

Алгоритм для пошуку зображень за текстом:

1) На тестовому наборі даних з зображеннями обраховуються вектори ознак;
2) Вводиться номер, створюється синтетичне зображення з нього, яке також перетворюється у вектор ознак;
3) Для кожної пари таких векторів ознак обраховується косинусна схожість;
4) Обраховані відстані сортуються у порядку спадання, і обираються три зображення, найближчих до синтетичного відносно косинусної схожості;
5) Зображується синтетичне зображення із тексту (Original) і три найближчі до нього (Similar1, Similar2, Similar3)

Приклади синтетичних і знайдених схожих зображень на рисунках нижче: перше - для номеру за пошуком “АА5149ЕК”, друге - “АА5911ІН”

Рис.4.6 - візуалізація результату роботи програми

Як можна побачити з прикладів, пошук зображень відбувається вдало, проте точність пошуку необхідно оцінити, що є простою задачею, адже у нас є тільки індекси синтетичних і реальних зображень, за якими буде відбуватися порівняння.

4.5 Оцінка роботи алгоритма

4.5.1 Top-1 accuracy

Для оцінки роботи алгоритму будемо використовувати дві метрики: Top-1 accuracy і Top-3 accuracy. Ці метрики дозволять оцінити точність у контексті схожості синтетичного зображення до одного і трьох реальних зображень відповідно. Це міра ефективності, яка використовується в основному в задачах класифікації для оцінки моделей машинного навчання. Вона вимірює частку випадків, коли найкращий прогноз моделі є правильним. У нашому випадку, це топ-1 найближчий вектор. Обчислюється за такою формулою:

$$Top - 1\,Accuracy = \frac{Correct\ top{-}1\ predictions}{Number\ of\ Predictions}$$

Отже, необхідно знайти для кожної текстової репрезентації найближчі вектори з набору реальних зображень і оцінити точність за допомогою цих метрик.

4.5.2 Top-3 accuracy

Оцінка за цією метрикою є допоміжним сигналом з аналізу якості. Очевидно, за умови, якщо необхідне зображення не потрапить у топ-1, воно може потрапити у топ-3 схожих, а це підвищує рівень оцінки. Таким чином, виконавши пошук трьох зображень можна зрозуміти глибше, де алгоритм може допускатися помилки.

4.5.3 Обчислення Top Accuracy

Для обчислення двох вище згаданих метрик, використовувався такий підхід- спочатку, для реальних зображень генеруються векторні представлення за допомогою функції $autoencoder.predict()$. Потім- для кожного синтетичного зображення генеруються такі самі векторні представлення за допомогою тієї ж функції. Далі виконується алгоритм пошуку схожих зображень, де використовується косинусна схожість.

Яким чином можна порахувати кількість правильно знайдених зображень? Єдиним можливим способом є тільки за індексами зображень. Оскільки індекси синтетичних зображень збігаються з індексами відповідних реальних зображень, то таким чином рахується відсоток збігів за індексами, що відповідає top-N accuracy.

Для top-1 accuracy знаходимо відсоток правильних перших збігів за індексами. Наприклад, синтетичне зображення для тексту “05-438PB” має індекси [1] та [2], отже, якщо для синтетичного зображення для відповідного тексту знаходиться один із відповідних індексів, це враховується як 1, інакше 0.

Метрика	Top-1	Top-3
Точність	71.4	87.3

4.6 Переваги запропонованого алгоритму

Запропонований алгоритм має декілька значущих переваг. По- перше, це здатність працювати на не розмічених даних, тобто, для його використання у прикладних задачах необхідний тільки набір зображень, на якому відбуватиметься пошук і текст номерного знака, який шукається. По- друге, це можливість зберігати дані у векторній формі, що буде економити обчислювальні потужності, тобто, вхідний набір даних можна відразу перетворити у вектори і працювати виключно з ними.

Висновки

В даній роботі було розглянуто використані архітектури нейронних мереж, які використовувалися для імплементації алгоритму пошуку номерних знаків за текстовим запитом. Архітектури згорткових мереж, автокодувальника і сіамської мережі широко використовуються у завданнях Image Search та Image Similarity. Враховуючи їх властивості, стало можливо запровадити алгоритм, що враховує спільні властивості для зображень з однаковим номерним знаком. Це відбувається завдяки навчанню нейронної мережі, яка обраховує вектори ознак, які для схожих зображень мають малу відстань, а для не схожих зображень- велику. Така відстань обраховувалася за допомогою косиноснуї схожості або евклідової відстані.

Проаналізовано популярні алгоритми з розпізнавання номерів, їх переваги і недоліки. Виявлено, що наявні відомі алгоритми не здатні виконувати пошук за запитом, оскільки вони не мають можливості до асоціювання схожих зображень. Такі алгоритми використовуються для одностороннього розпізнавання тексту на зображеннях і не працюють в іншу сторону, тобто не шукають зображення за текстом.

Запропоновано власний алгоритм з розпізнавання і пошуку номерних знаків, що базується на архітектурі автокодувальника і новій ідеї генерування синтетичних зображень для роботи виключно у площині зображень, відійшовши від роботи з текстом і зображенням одночасно. Маючи два набори розмічених зображень, на першому була протестована можливість алгоритму працювати у такому форматі (знаходити спільні характеристики для пари зображень), а другий використовувався для остаточного навчання і валідації. Результати навчання показали, що точні зображення з набору даних для запитів знаходяться правильно у 71.4% випадків. Цей показник буде покращуватися при більшому наборі зображень для тренування і при поглибленні або удосконаленні архітектури автокодувальника. Оскільки було використано ванільний автокодувальник, імовірно, варіаційний покращить якість навчання.

Недоліками запропонованого алгоритму є велика кількість зображень, які не завжди на початковому етапі обробляються коректно, тобто деякі символи можуть або зникнути або зробитися розпливчастими, що впливає на якість навчання.

Можливості для покращення алгоритму

Як згадувалось раніше, є кілька змінних, які зможуть підняти точність. По-перше, це збільшення набору даних до близько 50 тисяч зображень, оскільки з об’ємами інформації працюють не всі платформи. По-друге, це покращення архітектури автокодувальника. Архітектуру можна зробити глибшою, додати штрафуючі шари, або використати інший вид автокодувальників, такі як варіаційний. Проте, впровадження цих кроків матиме кратне збільшення часу для тренування, оскільки на базовій архітектурі навчання тривало близько двох годин.

Список використаних джерел

1. How convolution neural networks interpret images // Режим доступу: https://morioh.com/a/c944e8e7ee9f/how-convolution-neural-networks-interpret-images#google_vignette
2. Afridi, Tariq & Alam, Aftab & Khan, Numan & Khan, Jawad. (2020). A Multimodal Memes Classification: A Survey and Open Research Issues.

3. Unveiling the Power of Autoencoders // Режим доступу: https://medium.com/@ompramod9921/unveiling-the-power-of-autoencoders-b2834d8743ae
4. KMeans as a classifier for the WIFI and MNIST datasets // Режим доступу: https://linux-blog.anracom.com/2022/05/06/kmeans-as-a-classifier-for-the-wifi-and-mnist-datasets-v-cluster-based-classification-of-the-mnist-dataset/
5. Siamese Neural Network(SNN) // Режим доступу: https://tmleyncodes.hashnode.dev/siamese-neural-networksnn
6. Haque, Naimul & Islam, Samira & Tithy, Rabeya & Uddin, Mohammad Shorif. (2023). Automatic Bangla License Plate Recognition System for Low-Resolution Images. 10.1109/STI56238.2022.10103289.
7. Shashirangana, Jithmi & Padmasiri, Heshan & Meedeniya, Dulani & Perera, Charith. (2020). Automated License Plate Recognition: A Survey on Methods and Techniques. IEEE Access. 9. 11203-11225. 10.1109/ACCESS.2020.3047929.
8. CLIP: Connecting text and images // Режим доступу: https://openai.com/index/clip/
9. Denoising MNIST images using an Autoencoder // Режим доступу: https://medium.com/@connectwithghosh/denoising-images-using-an-autoencoder-using-tensorflow-in-python-1e2e62932837
10. EasyOCR: A Comprehensive Guide // Режим доступу: https://medium.com/@adityamahajan.work/easyocr-a-comprehensive-guide-5ff1cb850168
11. Franco, Edian & Rana, Pratip & Cruz, Aline & Calderón, Víctor & Azevedo, Vasco & Ghosh, Preetam & Ramos, Rommel. (2021). Performance Comparison of Deep Learning Autoencoders for Cancer Subtype Detection Using Multi-Omics Data. 10.20944/preprints202102.0365.v1.

12. Li, Yikai & Chen, C. & Zhang, Tong. (2022). A Survey on Siamese Network: Methodologies, Applications and Opportunities. IEEE Transactions on Artificial Intelligence. PP. 1-21. 10.1109/TAI.2022.3207112.

13. Chang, Shyang-Lih & Chen, Li-Shien & Chung, Yun-Chung & Chen, Sei-Wang. (2004) Automatic License Plate Recognition. Intelligent Transportation Systems, IEEE Transactions on. 5. 42 - 53. 10.1109/TITS.2004.825086.

14. Hasnat, Md & Nakib, Amir. (2021) Robust License Plate Signatures Matching Based on Multi-Task Learning Approach. Neurocomputing. 440. 10.1016/j.neucom.2020.12.102.

15. Marzuki, P. & Radzi, Feeza & Wong, Yan Chiew & Abdul Hamid, Norihan & Ali, Nur & Mat ibrahim, Masrullizam. (2019) A design of license plate recognition system using convolutional neural network. International Journal of Electrical and Computer Engineering (IJECE). 9. 2196. 10.11591/ijece.v9i3.pp2196-2204

16. Omar, Naaman & Zeebaree, Subhi & M.Sadeeq, Mohammed & Zebari, Rizgar & Shukur, Hanan & Alkhayyat, Ahmed & Haji, Lailan & Kak, Shakir. (2023). License plate detection and recognition: A study of review. AIP Conference Proceedings. 050045. 10.1063/5.0170932.

17. Lars Lien Ankile, Morgan Feet Heggland, Kjartan Krange (2005). Deep Convolutional Neural Networks: A survey of the foundations, selected improvements, and some current applications. arXiv:2011.12960

18. A Review of the Autoencoder and Its Variants: A Comparative Perspective from Target Recognition in Synthetic-Aperture Radar Images. IEEE Geoscience and Remote Sensing Magazine. 6. 44-68. 10.1109/MGRS.2018.2853555.

Лістинг до програмного коду

1) **Генерація синтетичних зображень**

```python
for img_name in os.listdir ("ukr_LP"):
      img = cv2.imread("ukr_LP/"+img_name)
      license_plate_crop_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
      gray1 = cv2.bilateralFilter(license_plate_crop_gray, 13, 5, 15)
      ret,thresh1 = cv2.threshold(license_plate_crop_gray ,0, 255,cv2.THRESH_OTSU|cv2.THRESH_BINARY)
      thresh1 = cv2.resize(thresh1,(512,64))
      height, width = thresh1.shape
      vertical_pixel_sum = np.sum(thresh1, axis=0)
      myprojection = vertical_pixel_sum / 255
      blankImage_car_plate = np.zeros_like(thresh1)
      for i, value in enumerate(myprojection):
            final_hist = cv2.line(blankImage_car_plate, (i, 0), (i,int(value)), (255, 255, 255), 1)
      print(img_name," ", y)
      plt.imshow(final_hist)
      image_car_plates.append(thresh1)
      image_hists.append(final_hist)
      y+=1



      string = img_name
      result_array = []
      for char in string:
          if char.isdigit():
              result_array.append(char)
          else:
              result_array.append(char)
      down = result_array.index('_')
      del result_array[down:]
      result_array = [letter.lower() for letter in result_array]
      car_plate = np.zeros((50,50))
      for i in range(len(result_array)):
        if i ==0:
            letter = result_array[i]
            path = 'ALPHABET/'+letter+'.png'
            target_img = cv2.cvtColor(cv2.resize(cv2.imread(path),(64,64)),cv2.COLOR_BGR2GRAY)
            car_plate = target_img
        if i>0:
            if result_array[i]=="-":
                letter='--'
                path = 'ALPHABET/'+letter+'.png'
                target_img = cv2.cvtColor(cv2.resize(cv2.imread(path),(64,64)),cv2.COLOR_BGR2GRAY)
                car_plate = cv2.hconcat([car_plate,target_img])
            else:
                letter = result_array[i]
                path = 'ALPHABET/'+letter+'.png'
                target_img = cv2.cvtColor(cv2.resize(cv2.imread(path),(64,64)),cv2.COLOR_BGR2GRAY)
                car_plate = cv2.hconcat([car_plate,target_img])


      ret,threshN = cv2.threshold(car_plate ,0, 255,cv2.THRESH_OTSU|cv2.THRESH_BINARY)
      height, width = threshN.shape
      #string_hists.append(np.array(car_plate))
      vertical_pixel_sum = np.sum(threshN, axis=0)
      myprojection = vertical_pixel_sum / 255
      blankImage_car_plate2 = np.zeros_like(threshN)
```

2) Тренування автокодувальника на сервері з GPU

```python
with tf.device('/device:GPU:0'):
  input_img = keras.Input(shape=(64, 512, 1))

  x = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same')(input_img)
  x = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
  x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
  x = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
  x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
  x = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
  x = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
  x = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
  x = tf.keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
  encoded = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)



  x = tf.keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
  x = tf.keras.layers.UpSampling2D((2, 2))(x)
  x = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',padding = 'same')(x)
  x = tf.keras.layers.UpSampling2D((2, 2))(x)
  x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
  x = tf.keras.layers.UpSampling2D((2, 2))(x)
  x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
  x = tf.keras.layers.UpSampling2D((2, 2))(x)
  x = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
  x = tf.keras.layers.UpSampling2D((2, 2))(x)
  decoded = tf.keras.layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

  autoencoder = keras.Model(input_img, decoded)
  #opt = keras.optimizers.Adam(learning_rate=0.01)
  autoencoder.compile(optimizer='adam', loss='mse',metrics=['accuracy'])
  autoencoder.load_weights('model_weights.h5')
```